INTEGRATION OF PLANNING AND EXECUTION: FINAL-ASSEMBLY SEQUENCING

Hanan Luss, Moshe B. Rosenwein, and Elizabeth T. Wahls

Hanan Luss *is a supervisor in the Operations Research Department, and* ***Moshe B. Rosenwein*** *is a member of technical staff in that department. They are with AT&T Bell Laboratories in Holmdel, New Jersey. The work reported here was done while* ***Elizabeth T. Wahls*** *was a member of the Operations Research Department. She is now an information systems staff member in the Information Automation Engineering organization with AT&T Network Systems at AT&T's Merrimack Valley Works in North Andover, Massachusetts. Mr. Luss is responsible for research on operations research methodology and for related applied work, which currently emphasizes manufacturing, distribution, and transportation problems. He joined AT&T in 1973 and has both a B.Sc. and an M.Sc. from the Technion—Israel Institute of Technology, Haifa, and a* (continued on page 109)

The manufacture of complex telecommunications systems typically involves a final-assembly shop that assembles a variety of highly customized end products, each requiring many parts. Under a just-in-time (JIT) manufacturing discipline, final assembly *pulls* the production of parts from multiple feeder shops. The demands imposed on feeder shops are determined by the sequence of end products assembled in the final-assembly shop. To maintain a JIT discipline, such demands must be smooth over time. We describe the model and algorithm that form the basis of the final-assembly sequencer (FAS), a software tool that smooths the demands on each feeder shop. The final-assembly sequencing algorithm uses an efficient search heuristic to select a nearly optimal sequence from among many feasible alternatives. AT&T's Denver Works has implemented FAS as part of the integrated pull manufacturing (IPM) system for producing private-branch exchange (PBX) systems. Versions of FAS have also been implemented at other AT&T manufacturing locations.

Introduction

The Denver Works manufacturing environment (Figure 1) consists of multiple feeder shops that supply a variety of equipment subassemblies (e.g., cabinets, carriers, and power units) and different types of circuit packs to a flexible final-assembly operation. (We will use *parts* to mean the equipment subassemblies *and* circuit packs.) The primary role of the Denver Works is the manufacture of PBX systems, which is discussed by Carboy et al. in this issue.[1]

Just-In-Time Manufacturing for PBX Systems. Under JIT manufacturing, the final-assembly sequence determines the quantities of parts to be produced over time in the feeder shops. (*Final-assembly sequence* refers to the order in which end products are to be assembled in the

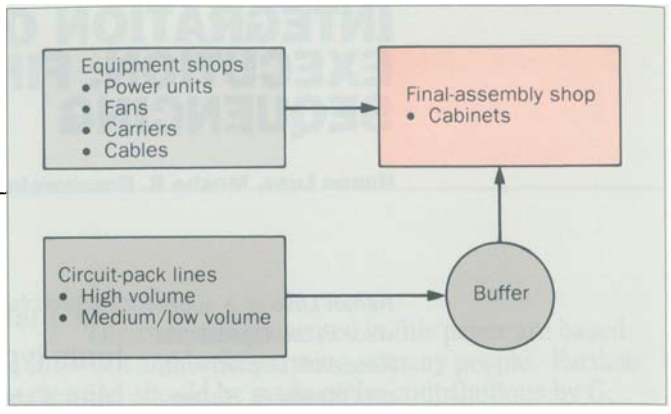

Figure 1. Feeder shops supply equipment subassemblies and circuit packs for final assembly into cabinets. A manual kanban system controls the flow of subassemblies from the equipment shops. A final-assembly sequence, lot sizes, and lead times determine when manufacture should start for particular circuit packs. If the demands for circuit packs are smooth over time, only a small buffer of finished circuit packs will accumulate.

final-assembly shop. *End product* refers to a completed product, i.e., a PBX system, comprised of multiple parts.) Ideally, the parts required for a particular end product should arrive simultaneously at the final-assembly shop from different feeder shops; i.e., feeder-shop production should be synchronized with the final-assembly sequence. If the end products consist of a mix of product types or are custom made, then each end product may require varying quantities of many parts. Therefore, the final-assembly sequence affects the demands imposed over time on the feeder shops. An effective sequence must impose smooth demands for the parts that these shops manufacture.

Under JIT, several methods exist to control production in the feeder shops, given a final-assembly sequence. In the PBX application, equipment subassemblies are characterized by low to medium levels of optionality (i.e., only a few designs and features may be offered) and relatively simple technology. These shops have shorter production intervals and use a limited set of components. (*Production interval* is the amount of time required for a part to traverse a feeder shop from start to finish.) Thus, a kanban system[2] easily controls the flow of equipment subassemblies to final assembly. In a kanban system, the quantity of subassemblies started in production in a particular time interval is equal to the quantity withdrawn by the final-assembly shop during the same interval. Each kanban card represents a part. When subassemblies are consumed in final assembly, their kanban cards serve as replenishment orders to the equipment shops.

In contrast, circuit packs exhibit high levels of optionality (i.e., many choices for components, features, and capability) and complex technology. A given end product may use many different circuit-pack types, each with a different production interval. Kanban systems, therefore, are inappropriate. (See Doshi and Krupka in this issue.[3]) Instead, the quantity of circuit packs started in production during each time interval is determined from the average production interval of the circuit packs, the lot-sizing rules, and the final-assembly sequence. (A *lot* is a batch of parts of the same type being moved together from one work cell to another.) Each circuit-pack type is manufactured in a small lot, and the lot size does not necessarily equal the quantity needed by a particular end product.

If the final-assembly shop imposes smooth demands over time, then only a small buffer (Figure 1) of finished circuit packs will accumulate. In contrast, high variabilities in demand may require the use of larger circuit-pack buffers to avoid significant variabilities in the feeder-shop production rates and delays in the final-assembly area.

Final-Assembly Sequencing for JIT Manufacturing. We describe a model and algorithm for efficient final-assembly sequencing in a JIT environment. The model smooths the requirements for each part over time. Ideally, over any interval of fixed length, referred to as the *smoothing interval*, the demand for a particular part should be (roughly) constant. For example, the smoothing interval for a particular part can be set equal to the part's production interval. In that case, the smoothing interval links the final-assembly sequencing problem with smoothing the work in process (WIP) of parts in the feeder shops.

The model also considers schedule constraints,

such as the product-dependent earliest-release and due dates. (*Earliest-release date* refers to the earliest date the end product could be assembled, and *due date* is the customer-imposed delivery date.) The complexity of the final-assembly sequencing problem stems from the large number of end products to be sequenced and the differing usage patterns of various parts in each end product.

The final-assembly sequencing model is formulated as an optimization problem. The model's objective function seeks to smooth the demands for each part made in the feeder shops. At each iteration, the final-assembly sequencing algorithm attempts to smooth the demands for the part that has the most variable usage pattern (this is recomputed at each iteration), without significantly increasing the imbalances in the usage patterns of other parts.

The algorithm uses an efficient local-search heuristic, based on selective pairwise interchanges, to solve problems of realistic size in reasonable running times. Mathematical details of the algorithm are given in Groeflin et al.[4]

Alternative models consider an aggregate measure of smoothness over all parts without explicitly considering the smoothness of individual parts. Monden gives[2] the first treatment of final-assembly sequencing, in the context of the Toyota Production System. His model minimizes an aggregate measure of smoothness. In his solution approach, end products are added to the sequence one by one. Recent papers[5-7] have extended Monden's model and solution method.

The software implementation of our algorithm is known as the *final-assembly sequencer* or FAS. A version of FAS has been implemented within the IPM system[8] at AT&T's Denver Works to sequence the final assembly of PBX systems. The differing levels of optionality and demand volatility of the subassemblies, coupled with management's desire for the factory to operate according to pull manufacturing, imply that FAS is an appropriate technology for the Denver Works.

In the next section, we describe the final-assembly sequencing model. Then, we describe the algorithm and also provide empirical validation of the model and algorithm. The final sections of the paper discuss the implementation of FAS.

The Final-Assembly Sequencing Model

The final-assembly sequencing problem is concerned with sequencing end products, e.g., a weekly "bucket" of customer demand, to minimize the part-usage variability simultaneously for all parts required for the final assemblies. Figure 2a displays multiple parts delivered to a final-assembly shop. This shop plans to assemble n end products during the schedule horizon, e.g., one week. As depicted by the varying heights of the rectangles in Figure 2a, each end product consists of a varying amount of m different parts produced in the feeder shops. For example, end product 1 requires a large amount of part 1, whereas end product 2 requires a small amount of part 1. The schedule horizon consists of n time slots—each of identical duration—that represent end-product assembly time in the final-assembly shop.

Figure 2b displays the usage of part 1 for the same sequence of end products as Figure 2a. (For clarity, the heights of the rectangles that correspond to the demands for part 1 have been proportionally scaled upward.) The depicted sequence of final assembly shows a pattern that alternates between end products that require large quantities of the part and end products that require small quantities of the part.

An intuitive measure of smoothness implies that, over any interval of fixed length, the demand for a part is roughly equal. Suppose that the smoothing interval is equal to the demand associated with two consecutive end products. Then, the sequence in Figure 2b imposes relatively smooth demands for part 1, because the sum of the demands for any two consecutive end products is roughly identical. If we interchanged the positions of end products 2 and 5 in the sequence, then three end products (1, 5, and 3) with large demands for part 1 would be assembled consecutively, thus creating imbalances in

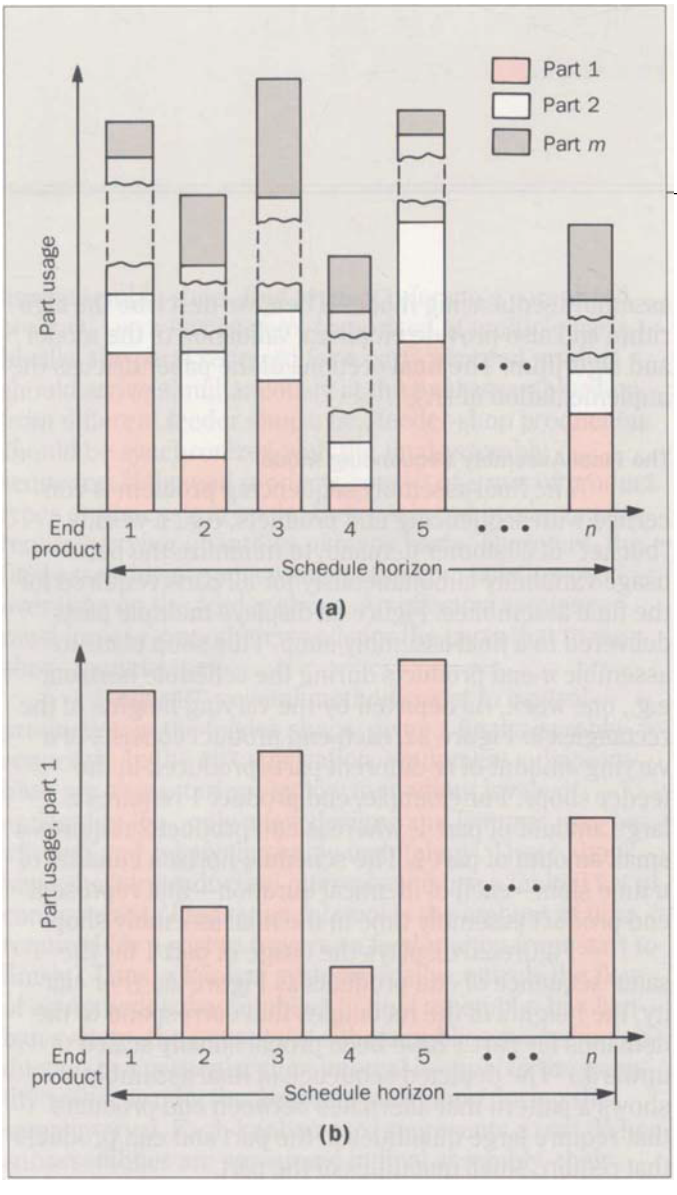

Figure 2. The final-assembly sequencing model tries to minimize the usage variabilities of parts required for final assembly and thus impose smooth demand for parts on the feeder shops. The schedule horizon consists of *n* time slots of equal duration, each representing the assembly of one end product. (a) The *n* end products to be assembled in this sequence require varying amounts of each part. (b) The usage pattern for part 1 for the same sequence of end products shows alternating high and low demand for the part.

demand over time. Obviously, a sequence that provides relatively smooth demands simultaneously for each of the m different parts is difficult to find.

Developing the Model. To develop the final-assembly sequencing model, we define the following concepts. Let $S = \{s(1), s(2), \cdots, s(n)\}$ be the sequence of end products assembled in the final-assembly shop. For example, $s(1) = 10$ and $s(2) = 78$ imply that end product 10 is assembled first and end product 78 is assembled second. Define r_{ij} as the amount of part i required for end product j.

To define smoothness formally, we need to define a smoothing interval, denoted as t_i, for part i. Ideally, the demand for each part i over any t_i consecutive time slots is kept roughly at the same level.

In our model, each time slot corresponds to the assembly of an end product. Suppose a work shift corresponds to a smoothing interval. Thus, if 80 end products are assembled each week and a plant operates 10 shifts per week, then 8 would be the value for the smoothing interval. The demand for part i over t_i consecutive time slots is equal to the sum of the requirements for part i in t_i consecutive end products in the sequence S.

For each part i, we define q_{ik} as the demand for part i over the t_i consecutive time slots, beginning with time slot k. That is, q_{ik} is the sum of demands for part i imposed by end products $s(k), s(k+1), \cdots, s(k+t_i-1)$:

$$q_{ik} = \sum_{h=k}^{k+t_i-1} r_{i,s(h)} \; . \qquad (1)$$

In the example above, suppose that $r_{1,10} = 5$, $r_{1,78} = 3$, and $t_1 = 2$. Then, $q_{11} = 5 + 3 = 8$. To treat the sum of demands properly for time slots at the end of the horizon, the model assumes cyclic demand. That is, if $n = 80$, then we assume that $r_{1,s(81)} = r_{1,s(1)}$, $r_{1,s(82)} = r_{1,s(2)}$, and so on.

We proceed to formalize the concept of smooth-

ness. Ideally, for any part i, q_{ik} is identical for all k. But because of the varying requirements for parts that stem from the optionality of the end products, we are unlikely to obtain a perfectly smooth part-usage profile with any sequence.

The measure of smoothness considered by our model is the largest q_{ik} for each part i, denoted by q_i,

$$q_i = \underset{k=1,2,\cdots,n}{\text{maximum}} \; q_{ik} \; . \tag{2}$$

A lower bound for q_i, denoted by $\bar{q}_i$, is the average demand for part i multiplied by its smoothing interval t_i, namely,

$$\bar{q}_i = t_i \frac{\sum_{j=1}^{n} r_{ij}}{n} \; . \tag{3}$$

Certain factors may affect the importance of smoothing the demands for individual parts. These include, for example, a part's cost or the utilization rate of its corresponding feeder shop. To reflect the relative importance of smoothing part i, we introduce a weight w_i. We may thus define a weighted deviation from the lower bound as $w_i(q_i - \bar{q}_i)$. The weighted deviations are normalized to capture variability in demand rates for individual parts so that we may compare high- and low-volume parts. Thus, we define:

$$v_i = \frac{w_i(q_i - \bar{q}_i)}{\bar{q}_i} \; , \quad \text{for } i = 1, 2, \cdots, m \tag{4}$$

as the weighted, normalized deviation for part i. In the remainder of the paper, we will refer to the weighted, normalized deviations as simply *the deviations*. (If normalization is not desirable, we can multiply w_i by $\bar{q}_i$ for every part i.)

An objective function that minimizes the sum of the deviations over all parts may lead to large deviations for certain parts. We strive to reduce the deviation of each part, giving priority to smoothing parts with larger deviations. If the deviations are sorted in nonincreasing order, then a desirable final-assembly sequence minimizes the value of the first (i.e., the top) element of the list; next, it minimizes the second-element value without increasing the first-element value; and so on.

A *minimax operator* minimizes the largest deviation v_i over all parts i. But to minimize the succeeding largest deviations (i.e., the second, third, etc.), we need to introduce a *lexicographic operator*. Let:

$$\mathbf{v} = (v_{a(1)}, v_{a(2)}, \cdots, v_{a(m)})$$

denote a vector of v_is, sorted in nonincreasing order. That is, $v_{a(1)}$ is the largest of all the v_is, $v_{a(2)}$ is the second largest, etc. The vector $\mathbf{v}^1$ is lexicographically smaller than the vector $\mathbf{v}^2$, if $v^1_{a(h)} < v^2_{a(h)}$ in the first element where the two vectors differ, e.g., in the hth element. For example, if $\mathbf{v}^1 = (v^1_3, v^1_1, v^1_4, v^1_2) = (10, 7, 5, 3)$ and $\mathbf{v}^2 = (v^2_4, v^2_2, v^2_3, v^2_1) = (10, 7, 6, 1)$, then $\mathbf{v}^1$ is lexicographically smaller because $v^1_3 = v^2_4 = 10$, $v^1_1 = v^2_2 = 7$, and $v^1_4 = 5 < 6 = v^2_3$.

In constructing a sequence S, the model must also consider the earliest-release date and due date associated with each end product. An end product may not be assembled before its earliest-release date (e.g., because material was not available) and must be assembled no later than its customer-imposed delivery date. In the model, the constraints on earliest-release date and due date are translated into time slots in the schedule horizon. For example, a due date of 10 implies that the corresponding end product must be among the first 10 elements of the sequence S. Panel 2 states the model.

Production Interval. The smoothing interval for any part i can be related to the part's production interval, denoted by τ_i.

Consider a replenishment system, implemented with kanban cards, where the quantity of part i that starts

Panel 2. Model Statement

For any sequence S, let the deviations v_i (for $i = 1, 2, \cdots, m$) be computed by equations (1) through (4). Find a sequence $S = \{s(1), s(2), \cdots, s(n)\}$ that lexicographically minimizes the nonincreasing sorted vector $\mathbf{v} = (v_{a(1)}, v_{a(2)}, \cdots, v_{a(m)})$, subject to satisfying earliest-release date and due date constraints.

in production in a feeder shop during a particular time slot is equal to the quantity of part i consumed by the final-assembly shop during the same time slot. Thus, at any point in time, the WIP of part i in the feeder shop is equal to the sum of the demands imposed by τ_i end products, assembled consecutively. If we let $t_i = \tau_i$, then minimizing v_i is equivalent to minimizing the weighted WIP deviation for part i. Luss studied[9] the implication of final-assembly sequencing on the WIP and required buffer sizes under various manufacturing disciplines.

Final-Assembly Sequencing Algorithm

Final-assembly sequencing is a complex combinatorial problem. We developed a heuristic that carefully considers a very small fraction of sequences from which it selects a good sequence. The heart of the algorithm is an efficient interchange heuristic that attempts to swap the order of assembly of a pair of end products. An interchange procedure is applied repeatedly to an initial feasible solution until no further exchanges improve the objective. Mathematical details are provided in Groeflin et al.[4]

The final-assembly sequencing algorithm consists of three modules: preprocessing, feasibility, and optimization.

The *preprocessing module* reduces the size of the problem by eliminating any part whose usage is constant across end products. Also, if two (or more) parts have proportional usage patterns and identical smoothing-interval parameters, then all but one part (the one that corresponds to the largest weight w_i) can be eliminated from the smoothing problem. Preprocessing shrinks the number of parts that the algorithm needs to smooth.

Panel 3. Feasibility Module

Step 0—Initialization. All end products are unsequenced. *Point* to the first position in S and label it as the *current* position.

Step 1—Termination test. If there exists an unsequenced end product whose due date (in terms of position within S) is less than the current position, then stop. Such an end product cannot be feasibly sequenced.

Step 2—Selection. Among all unsequenced end products, choose an end product whose earliest-release date is equal to or smaller than the current position and whose due date is the minimum of all such unsequenced end products. Ties are broken arbitrarily. Assign the selected end product to the current position in S. If no such end product exists, a feasible sequence does not exist; therefore, terminate the module.

Step 3—Update pointer. If the current position is the last position in S, then terminate the module with an initial feasible sequence. Else, move the pointer to the next position in S, and relabel it as the current position. Go to Step 1.

The *feasibility module* finds an initial feasible sequence that satisfies the earliest-release date and due date constraints (specified in terms of the end-product positions in S). The presence of earliest-release date and due date constraints implies that a feasible sequence may not exist. For example, if six end products have an earliest-release date of 11 and a due date of 15, then a feasible sequence does not exist. If a feasible sequence is not encountered, the final-assembly sequencing algorithm terminates. Obviously, if each end product's earliest-release date corresponds to the beginning of the schedule horizon (i.e., time slot 1) and each end product's due date

corresponds to the end of the schedule horizon (i.e., time slot n), then any arrangement of the end products is a feasible initial sequence. Panel 3 gives details of the module that finds an initial feasible sequence.

The *optimization module* consists of an interchange procedure that attempts to minimize lexicographically the nonincreasing sorted vector of part-usage deviations v_i. For each proposed interchange, the vector of deviations **v** must be checked to ensure that the interchange improves the lexicographic objective. For instance, an interchange is acceptable if it reduces the deviation of a part that has a relatively large deviation, even if it increases the deviations of parts that have smaller deviations.

The description in Panel 4 provides a basic outline of the algorithm. For this paper, we have omitted numerous implementation details that expedite execution. The algorithm typically terminates after several passes through the list of deviations. Empirically, we observed that the bulk of the reduction of the deviations occurred on the first pass.

Example. We illustrate some of the points of the interchange procedure with a simple example.

Assume that two parts are to be assembled into five end products; i.e., $n = 5$ and $m = 2$. The smoothing interval for each part is 2, i.e., $t_1 = t_2 = 2$. Also, each part is weighted equally; i.e., $w_1 = w_2 = 1$. Each end product may be assembled at any position in S (no earliest-release and due date constraints are imposed). The part requirements for each end product are given in Panel 5A.

Suppose that the initial feasible sequence of final assembly is $\{1, 2, 3, 4, 5\}$. Then, the q_{ik}s (the sums of the requirements over two time slots for each part i) are given in Panel 5B. The values with asterisks correspond to q_i, the largest q_{ik} for each part i. Recall that the model assumes a cyclic pattern of demand; e.g., $r_{16} = r_{11}$, $r_{17} = r_{12}$, etc.

As stated before, the lower bound on each q_i is given by $\bar{q}_i$, the average requirements of part i per end product multiplied by t_i. Hence, $\bar{q}_1 = 28$ and $\bar{q}_2 = 24$. This implies that $v_1 = (32 - 28)/28 = 1/7$ and $v_2 = (30 - 24)/24 = 1/4$. Thus, part 2 is associated with the largest deviation in the sorted list of deviations; i.e., $v_{a(1)} = v_2 = 1/4$ and $v_{a(2)} = v_1 = 1/7$. We propose to interchange end products 2 and 4 in the sequence to obtain $S = \{1, 4, 3, 2, 5\}$. The q_{ik}s are then given in Panel 5C. Note that q_2 is reduced from 30 to 27, but q_1 remains unchanged at 32. The deviations are given as follows: $v_1 = (32 - 28)/28 = 1/7$, and $v_2 = (27 - 24)/24 = 1/8$. Part 1 is now associated with the largest deviation in the sorted list of deviations; i.e., $v_{a(1)} = v_1 = 1/7$ and $v_{a(2)} = v_2 = 1/8$.

Panel 4. Interchange Heuristic

Step 0—Initialization. Obtain an initial feasible final-assembly sequence. Sort the list of deviations that correspond to the different parts in nonincreasing order and obtain the vector:

$$\mathbf{v} = (v_{a(1)}, v_{a(2)}, \cdots, v_{a(m)}).$$

Let p be the index associated with a pointer; initially, $p = 1$.

Step 1—Propose interchange. Determine a previously untried interchange that reduces the deviation $v_{a(p)}$. If none exists and $p < m$, then increase p by 1 and repeat Step 1. If $p = m$, skip to Step 5.

Step 2—Evaluate interchange. Determine the effect of the proposed interchange that decreases $v_{a(p)}$ on deviations associated with all other parts. The proposed interchange is rejected if:

- $v_{a(h)}$ will increase for some integer $h < p$.
- for some $h > p$, $v_{a(h)}$ will increase so that its new value will exceed the value of $v_{a(p)}$ before the interchange.

Step 3—Execute interchange. If an interchange is rejected, return to Step 1. Else, if an interchange is not rejected in Step 2, then modify the sequence of final assembly and proceed to Step 4.

Step 4—Update the list. Sort the deviations again to obtain the newly ordered vector **v**. Return to Step 1 without changing the pointer's position (i.e., the pth position).

Step 5—Termination test. If all deviations remain unchanged on this pass through the entire list of parts, then stop. Otherwise, set $p = 1$, and go to Step 1.

Panel 5. Interchange Example

A. Parts requirements for the end products:

End product	1	2	3	4	5
Part 1	12	16	16	14	12
Part 2	12	9	9	15	15

Thus, $r_{11} = 12, r_{12} = 16, r_{21} = 12, r_{22} = 9$, etc.

B. Requirements q_{ik} over two time slots:

Time slot / End product	1 / 1	2 / 2	3 / 3	4 / 4	5 / 5
Part 1	28	32*	30	26	24
Part 2	21	18	24	30*	27

For example, $q_{11} = r_{11} + r_{12} = 12 + 16 = 28$, $\cdots$, $q_{15} = r_{15} + r_{11} = 12 + 12 = 24$.

C. End products 2 and 4 interchanged:

Time slot / End product	1 / 1	2 / 4	3 / 3	4 / 2	5 / 5
Part 1	26	30	32*	28	24
Part 2	27*	24	18	24	27

For example, $q_{11} = r_{1,s(1)} + r_{1,s(2)} = r_{11} + r_{14} = 12 + 14 = 26$.

D. End products 1 and 3 interchanged:

Time slot / End product	1 / 3	2 / 4	3 / 1	4 / 2	5 / 5
Part 1	30*	26	28	28	28
Part 2	24	27*	21	24	24

For example, $q_{11} = r_{1,s(1)} + r_{1,s(2)} = r_{13} + r_{14} = 16 + 14 = 30$.

Next, we propose to interchange end products 1 and 3 to obtain $S = \{3, 4, 1, 2, 5\}$. The results of this interchange appear in Panel 5D. Further attempts to reduce the deviation of either part are unsuccessful. Hence, at termination, $S = \{3, 4, 1, 2, 5\}$, which implies that $v_{a(1)} = v_2 = (27 - 24)/24 = 1/8$, and $v_{a(2)} = v_1 = (30 - 28)/28 = 1/14$.

Validating the Algorithm. Next, we describe some computational results. The final-assembly sequencing algorithm was programmed in Fortran 77, and computational testing was done on an Amdahl Model 3870 computer to validate the procedure empirically. (The computational speed of the Amdahl 3870 computer is comparable to that of the IBM Model 3090 computer, the mainframe computer used by Denver's production system.) From the Denver Works, we obtained historical data on customer orders for the Definity® System 75 telecommunications system. This data included the high-level bill of materials for each order; i.e., the list of major equipment subassemblies and circuit packs for each end product.

Table I gives the computational results from testing and validating the FAS software. The data in problem sets A, B, and C correspond to different scenarios of demand for PBX systems. We used the data to generate multiple problems, e.g., ten problems for problem set A. In developing a generic final-assembly sequencing algorithm, we experimented with problems of varying sizes. In the table, *number of parts* corresponds to the number after the preprocessing stage. Also, we assumed that all parts were weighted equally and that, for all end products, the earliest-release date was equal to 1 and the due date was equal to n. The t_is were randomly chosen to be uniform over the interval between 1/20 and 1/4 of n.

We compared the solutions of the final-assembly sequencing algorithm to the best solution among 50,000 randomly generated final-assembly sequences for each problem. The execution time for generating 50,000 random sequences was between 50 to 200 times greater than the average running times for the FAS software. FAS

running times are reasonable—on average, less than 20 seconds for a problem with 320 end products and 30 parts. Thus, these results suggest that FAS is suited to be run as part of a production-scheduling system and can also be used for *what if* sensitivity-analysis studies.

As Table I shows, FAS outperformed the best of the 50,000 random sequences in obtaining smooth demand schedules to be imposed on each of the feeder shops. For comparison purposes, we measured the smoothness of the demands imposed on all the feeder shops in aggregate, i.e., $\sum_{i=1}^{m} v_i$. Although the final-assembly sequencing algorithm does not try to minimize this metric, the aggregate measure it obtained was about one-third to one-half of the minimum aggregate measure obtained by the best of 50,000 random sequences.

FAS also outperformed the best random sequence on other measures of performance. For example, in 19 of the 20 problems, the vector of deviations that FAS generated was lexicographically smaller than the lexicographically smallest vector of deviations generated by 50,000 random sequences.

Table I. FAS versus 50,000 Random Sequences

	Problem set		
	A	B	C
Number of problems	10	6	4
Number of parts	20	25	30
Number of end products	80	160	320
Average CPU running time (seconds)			
FAS	1.38	3.13	18.41
50,000 random sequences	281.93	509.88	1051.21
Average sum of deviations; $\sum_{i=1}^{m} v_i$			
FAS	3.53	3.40	4.16
Best of 50,000 random sequences	6.45	8.41	12.35

Implementation of FAS at the Denver Works

A version of FAS has been installed at the Denver Works as part of the IPM system. FAS is used to sequence the final assembly of System 75 and System 85 PBXs and the AUDIX systems. (AUDIX stands for *audio information exchange.*) It simultaneously smooths the usage of equipment subassemblies and circuit packs. The success of the FAS tool is reflected in Denver's ability to operate according to JIT manufacturing principles.

FAS is run daily at the Denver Works, and each run generates a sequence for a schedule horizon of one week. In practice, only the first day of each schedule generated is ever executed. Denver needs the daily runs to accommodate recent changes in customer orders. Input to FAS is obtained from Denver's customer-order databases.

The final-assembly sequence must be passed to the feeder shops. As noted earlier, the equipment shops have lower manufacturing complexity and shorter production intervals. Thus, the final-assembly shop pulls equipment from the feeder shops via a manual kanban system. Circuit-pack manufacturing is more complex, and its production interval is longer (although greatly reduced since implementation of IPM and improvements in shop operations). FAS attempts to smooth the demand for each circuit-pack type over time. Other modules of IPM then determine the net requirements for each circuit-pack type by using the daily demand for circuit packs, as determined by FAS, and the circuit packs contained in WIP inventory. Production intervals for circuit packs and lot-sizing rules are used to schedule the production of circuit packs. Kinney, Denning, and Foo give[8] additional implementation details on IPM.

We also developed a graphics interface (using Lotus Corporation's 1-2-3® software) as a planning aid and for sensitivity analysis. The graphics system can be run interactively with the final-assembly sequencing

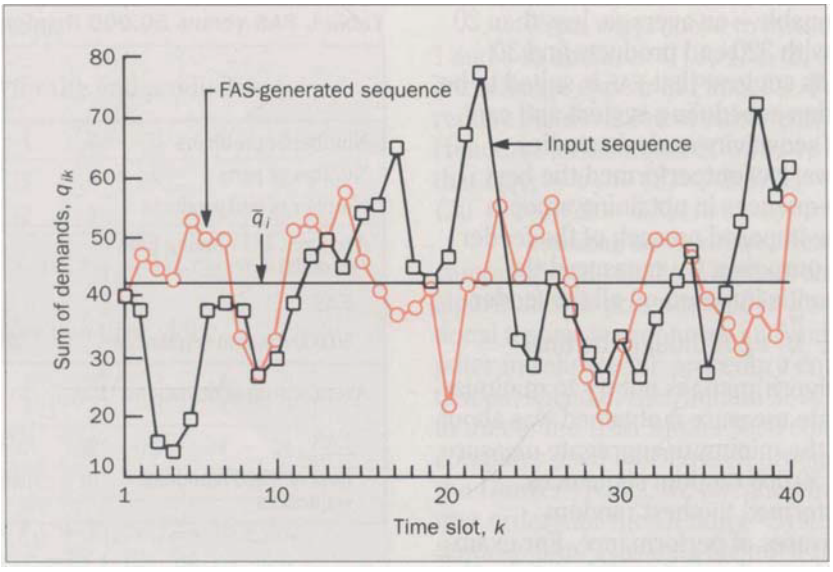

Figure 3. This graph compares two usage patterns for part *i*. q_{ik} is the sum of demands for part *i* over the smoothing interval that starts at time slot *k*. The curves show the q_{ik}s for part *i* based on a user-provided final-assembly sequence and a sequence generated by FAS. The FAS-generated sequence shows a significantly smaller value of q_i (the largest q_{ik}).

algorithm to fine-tune model parameters. For instance, initially, a user may set the weights for all the parts to one. Then, the weights can be adjusted to smooth the deviations of the more costly parts at the expense of causing more variable deviations of less expensive parts.

A user can display a graph of the smoothness of a part's usage pattern, i.e., a plot of the q_{ik}s versus k for any part i. The graph in Figure 3 compares a part's usage pattern obtained by a final-assembly sequence that a user provided, with the part's usage pattern obtained by the sequence generated by FAS.

Epilogue

FAS has been in use at the Denver Works since late 1987. It is an important module in the IPM system that is credited with reducing the production interval and finished-parts buffer associated with circuit-pack manufacturing. A new release of FAS, issued in 1989, has reduced the computational running time of the algorithm. For example, limiting the search for good interchanges to a subset of only those parts with the largest deviations reduced run times by up to 50 percent for problem set C in Table I, without noticeable degradation in the quality of the solution.

Because the final-assembly sequencing algorithm is a generic production-scheduling procedure for a pull manufacturing environment with a variety of customized end products, additional uses of the model and algorithm have arisen in AT&T manufacturing. FAS has been used at several locations, and others are exploring implementation. In one application, a slightly modified version of FAS efficiently generated final-assembly sequences for 600 end products that consume 15 different parts.

One location has also successfully implemented FAS as a front-end loading tool. If FAS is given a list of lots to be started in production and their usage of critical resources, it determines a start-work sequence of lots that creates a (roughly) uniform load on each resource. This production-scheduling application has spawned an extension of the final-assembly sequencing algorithm for front-end loading problems that explicitly considers the effects of machine set-up times. Other algorithms have been developed recently for front-end loading of flexible flow lines.[10–14]

Acknowledgments

Heinz Groeflin participated in the original development of the final-assembly sequencing model and algorithm. Adam N. Rosenberg assisted in preparing the 1989 release of FAS and also developed a version that considers set-up times. Terri L. Zehnder developed the computer interfaces between Denver's databases and FAS. We were also very fortunate to work with George Foo, Stanley A. Hendryx, John Huber, Lynn P. Jones, and John J. Svitak and with many people at the Denver Works.

References

1. J. D. Carboy, G. Foo, L. E. Kinney, L. P. Jones, and D. C. Krupka, "Striving for Manufacturing Excellence at the Denver Works: A Summary," *AT&T Technical Journal*, Vol. 69, No. 4, July/August 1990, pp. 5-18.
2. Y. Monden, *Toyota Production System*, Institute of Industrial Engineers, Industrial Engineering and Management Press, Norcross, Georgia, 1983.
3. B. T. Doshi and D. C. Krupka, "Integration of Planning and Execution: Theory and Concepts," *AT&T Technical Journal*, Vol. 69, No. 4, July/August 1990, pp. 90-98.
4. H. Groeflin, H. Luss, M. B. Rosenwein, and E. T. Wahls, "Final Assembly Sequencing for Just-in-Time Manufacturing," *International Journal of Production Research*, Vol. 27, No. 2, February 1989, pp. 199-213.
5. G. J. Miltenburg, "Level Schedules for Mixed Model Assembly Lines in Just-in-Time Production Systems," *Management Science*, Vol. 35, No. 2, February 1989, pp. 192-207.
6. G. J. Miltenburg and G. Sinnamon, "Scheduling Mixed-Model Multi-Level Just-in-Time Production Systems," *International Journal of Production Research*, Vol. 27, No. 9, September 1989, pp. 1487-1509.
7. R. R. Inman and R. L. Bulfin, "Sequencing JIT Mixed-Model Assembly Lines," *Management Science*, to be published.
8. L. E. Kinney, E. H. Denning, and G. Foo, "High-Option Manufacturing: The Denver Works Information-System Architecture," *AT&T Technical Journal*, Vol. 69, No. 4, July/August 1990, pp. 110-116.
9. H. Luss, "Synchronized Manufacturing at Final Assembly and Feeder Shops," *International Journal of Production Research*, Vol. 27, No. 8, August 1989, pp. 1413-1426.
10. R. J. Wittrock, "Scheduling Algorithms for Flexible Flow Lines," *IBM Journal of Research and Development*, Vol. 29, No. 4, July 1985, pp. 401-412.
11. R. J. Wittrock, "An Adaptable Scheduling Algorithm for Flexible Flow Lines," *Operations Research*, Vol. 36, No. 3, May-June 1988, pp. 445-453.
12. S. Kochar, R. J. T. Morris, and W. S. Wong, "The Local Search Approach to Flexible Flow Line Scheduling," *Proceedings of the 2nd International Conference on Production Systems*, Paris, France, April 6 to 10, 1987, INRIA (Institut National de Recherche en Informatique et en Automatique), Paris, 1987, pp. 175-186.
13. S. T. McCormick, M. L. Pinedo, S. Shenker, and B. Wolf, "Sequencing in an Assembly Line with Blocking to Minimize Cycle Time," *Operations Research*, Vol. 37, No. 6, November-December 1989, pp. 925-935.
14. P. K. Johri, "A Heuristic Algorithm for Loading New Work on Circuit Pack Assembly Lines," *International Journal of Production Research*, to be published.

Biographies (continued)

Ph.D. in operations research from the University of Pennsylvania, Philadelphia. Mr. Rosenwein's work focuses on research and applications in integer programming, manufacturing, and transportation. He joined AT&T in 1986 and has a B.S.E. from Princeton University, New Jersey, and an M.S. and Ph.D. in operations research from the University of Pennsylvania. Ms. Wahls works on the design and implementation of production-scheduling systems. She joined the company in 1985, and has a B.S. in applied mathematics from Brown University, Providence, Rhode Island, and an M.S. in operations research from Columbia University, New York.

(Manuscript received February 22, 1990)